PROYECTO FINAL DE CARRERA

TÍTULO DEL PFC: Joc 2D per Android per partides multijugador

TITULACIÓN: Ingeniería Técnica Informática de Sistemas

AUTOR: Francisco José Maroñas Bravo

DIRECTOR: Lluis Pérez Vidal

FECHA: 18 de Enero de 2011

Índice

1. Introducción

1.1 Oportunidades del proyecto

Actualmente nos encontramos en el apogeo de la era de los dispositivos móviles, más concretamente los teléfonos móviles conocidos como smartphones[1]. En esta categoría podemos clasificar un total de tres grandes sistemas operativos, Android, iOS[2] y Windows Phone[3].

Teniendo en cuenta la gran cantidad de posibilidades que nos ofrece este tipo de dispositivos, entre ellas las conexiones a internet, ya sea vía WI-FI o mediante una conexión de datos, surge la posibilidad de crear un juego multijugador para smartphones.

Forma parte de la vida cotidiana de la gran mayoría de nosotros, más concretamente de las personas que residen en grandes urbes, desplazarse en medios de transporte públicos, lo que genera unos espacios de tiempo libre, que cada vez es más habitual que la gente ocupe interactuando con sus teléfonos.

Esa interacción, dado el ritmo de vida ajetreado de este tipo de gente, suele reducirse a la consulta de redes sociales y a alejar su mente de sus problemas cotidianos, recurriendo en muchos casos a juegos sencillos pero adictivos.

Surge aquí entonces la oportunidad de unir ambos conceptos, la red social y el juego, en lo que a fin de cuentas sería este proyecto, un juego multijugador. Un juego que permita al usuario jugar con otros usuarios usando internet.

Esta unión nos permite conocer así dos ámbitos de la programación, el intercambio de datos mediante conexiones de datos, y el desarrollo de juegos.

[1] Teléfono móvil de gama alta construido sobre una plataforma de informática móvil, con capacidades de computación y conectividad avanzadas.
[2] iOS es un sistema operativo móvil de Apple, originalmente desarrollado para el iPhone.
[3] Sistema operativo móvil desarrollado por Microsoft para teléfonos con capacidades avanzadas.

Las conexiones de datos, nos permiten conocer más en profundidad la arquitectura del sistema Android, ya que requiere conocer la estructura del sistema y crear una interfaz de usuario amigable, que consiga que el usuario pueda iniciar su experiencia de juego de forma rápida y sencilla.

Y por otro lado, el desarrollo de videojuegos, nos permite conocer más aún el sistema, ya que es necesario conocer su funcionamiento para poder hacer un juego rápido, y por otro lado el diseño gráfico, que nos permita plantear un escenario y un ambiente de juego agradable.

En definitiva, este proyecto se interesa por conocer más en detalle el sistema operativo de google, y el desarrollo de videojuegos para plataformas móviles.

1.2 Objetivos

El objetivo primordial del proyecto que se plantea, es conocer la arquitectura del sistema operativo para dispositivos móviles desarrollado por Google, conocido comercialmente como Android. Para asumir este objetivo proponemos la creación de un juego multijugador que interactúe con un servidor en internet.

Este agrupa las dos partes más interesantes de Android, la comunicación del dispositivo mediante redes de datos hacia el exterior, y el desarrollo de una interfaz de usuario agradable que consiga que al usuario le sea sencillo iniciar su interacción con el juego.

La primera fase del proyecto consiste en preparar todo el entorno de desarrollo necesario para poder iniciar la construcción de nuestra aplicación. Este entorno se reduce el editor Eclipse acompañado de una serie de plugins[4] desarrollados por Google, como serían el ADT[5] y el AVD[6].

La segunda fase consiste en estudiar los diferentes frameworks[7] de programación de videojuegos OpenSource, que nos faciliten nuestra tarea a la hora de crear nuestra interfaz y dinámica de juego.

En una tercera fase, en la que ya tenemos todos los conocimientos necesarios para iniciar el desarrollo de nuestra aplicación, pasamos a implementar la versión final de este juego. El objetivo de esta fase es obtener un juego completamente funcional con el que el usuario pueda interactuar usando la librería AndEngine.

Asumimos una cuarta fase en la cual debemos integrar el juego con el servidor desarrollado para asumir el objetivo de hacer el juego multijugador, comunicándose por protocolos inalámbricos usando internet. El objetivo de esta fase es asumir la parte social del juego llegando así a un número mayor de usuarios.

[4] Aplicación que se relaciona con otra para aportarle nuevas funciones, generalmente muy específicas.
[5] Android Development Toolkit, plugin para el desarrollo de aplicaciones android en Eclipse.
[6] Android Virtual Device, emulador de dispositivos android destinado al desarrollo de aplicaciones.
[7] Estructura conceptual y tecnológica de soporte definido, normalmente con artefactos a módulos de software concretos, con base a la cual otro proyecto pueda ser más fácilmente organizado y desarrollado.

Por último tenemos la quinta fase, en la que la aplicación se verá sometida a una revisión, y a una batería de pruebas en diversos dispositivos, con el objetivo de encontrar y solventar el mayor número de bugs posible. Analizando finalmente las posibilidades de crecimiento que ofrece el desarrollo final llevado a cabo.

1.3 Estructura del documento

Este documento está dividido en un total de 7 apartados el contenido de cada uno de los cuales pasamos a detallar a continuación.

En el primer capítulo de este documento establecemos los objetivos y oportunidades de este proyecto, además de dar una pequeña introducción que nos ayuda a situarnos en el concepto del proyecto.

En un segundo capítulo se hace un estudio en profundidad del sistema operativo móvil Android. Con este fin, se estudian sus características, estructura y funcionalidades. Además hacemos un detallado de los componentes de una aplicación Android y de su funcionamiento.

En un tercer capítulo del documento hacemos un estudio profundo de las características del framework AndEngine y de cada una de las funcionalidades y extensiones que nos ofrece para llevar a cabo el desarrollo de nuestro juego.

Dentro del contenido del cuarto capítulo hacemos un repaso de la arquitectura del motor de físicas Box2D[8], del que se vale el framework AndEngine para poder simular todos los movimientos de nuestro juego de una manera lo más realista posible.

El quinto capítulo nos hace un repaso de la tecnología usada para la creación del servidor con el que vamos a interactuar y de la integración que hemos tenido que hacer en la aplicación para poder llevar a cabo esta interacción.

En este sexto capítulo se cubren las partes de las que se conforma la aplicación final, así como las clases, librerías y componentes usados para llevarla a cabo. Este capítulo nos permite tener una visión más detallada de como se ha llevado a cabo la aplicación y los resultados que finalmente se han obtenido.

En el séptimo capítulo se detallan las conclusiones y los objetivos asumidos una vez llevado a cabo el desarrollo de la aplicación, sin tener en cuenta su impacto social ni futuras ampliaciones.

[8] Biblioteca libre que implementa un motor de físicas de dos dimensiones, desarrollado en C++

Para finalizar el documento, en los capítulos séptimo y octavo, añadimos una bibliografía y una serie de anexos que detallan más una serie de conceptos.

2. Android

2.1 Filosofía

Tal y como hemos descrito en la introducción del proyecto, en el mercado de la telefonía móvil actual, hay una fuerte competencia entre compañías para demostrar que su sistema operativo es el más óptimo. Algunos de estos sistemas podrían ser iOS desarrollado por Apple, Windows Phone 7 desarrollado por Microsoft, Symbian[9], Bada desarrollado por Samsung, RIM[10]/QNX desarrollado por BlackBerry, y por su puesto el caso que nos ocupa, Android desarrollado por Google.

Android es un sistema operativo para teléfonos móviles basado en Linux, de hecho el núcleo del sistema operativo está basado en el kernel[11] 2.6 de este. El desarrollo de este sistema no fue iniciado por Google, sino que lo inició una empresa llamada Android Inc., que poco tiempo después despertó el interés de Google y se vio absorbida por esta, manteniendo, eso si el nombre de la idea originaria.

Actualmente el desarrollo de la plataforma Android, se lleva a cabo mediante la Open Handset Alliance, que es un consorcio creado en el 2007 con el objetivo común de desarrollar estándares para dispositivos móviles. Este consorcio está liderado por Google. El primer código que vio la luz a raíz de la creación de este consorcio, fue Android.

En octubre de 2008 se hace público el proyecto OpenSource conocido como Android, el sistema operativo para dispositivos móviles desarrollado en su mayor parte por Google, y que pasa a distribuirse bajo una licencia apache 2.0.

Junto con esta publicación se inicia la andadura del Android Market, plataforma que usa Android para la distribución de sus aplicaciones, y se lanza al mercado el primer terminal android comercial, conocido como HTC Dream o G1.

[9] Sistema operativo móvil nacido de la alianza de varias compañías móviles, nacido para competir con Windows Phone y Android
[10] Sistema operativo móvil desarrollado por Blackberry para entrar en el mercado de los smartphones.
[11] Principal responsable de facilitar a los distintos programas acceso seguro al hardware del ordenador y gestionar los recursos del sistema.

El éxito de Android se consolida en noviembre de 2009, cuando el Motorola Droid/Milestone, supera el récord de ventas del dispositivo iPhone, desarrollado por Apple, considerado su mayor competidor en el sector de los dispositivos móviles.

2.2 Características Principales

La principal característica del sistema operativo de Google es su optimización, todo su desarrollo se basa en la optimización de recursos por parte de este, con el objetivo de conseguir un sistema fluido en su funcionamiento sobre los dispositivos móviles y una buena optimización de batería.

La ejecución de aplicaciones se basa en el uso de la Máquina Virtual Dalvik[12], a la que se aplicó una optimización para su correcto funcionamiento en dispositivos móviles. Las aplicaciones que ejecuta el sistema están escritas en Java y empaquetadas en un formato conocido como Android Package (.apk).

Cada una de las aplicaciones del sistema es completamente independiente de todas las demás, lanzan cada una su propio proceso sobre el kernel linux sobre el que se ejecuta el sistema, cada una usa su propia máquina virtual Java con el fin de que el uso de diferentes aplicaciones no se interfiera entre sí.

Cada una de las aplicaciones que se ejecutan recibe su propio ID de usuario, al que se le asignan una serie de permisos, que se usan para poder acceder de manera única a los ficheros generados por la aplicación, garantizando así la seguridad de las diferentes aplicaciones y del sistema en general.

Formó parte del desarrollo del sistema la integración de un navegador web basado en WebKit, con el objetivo de hacer la experiencia de navegación del usuario más rápida, incorporando también un motor de optimización de páginas webs para el soporte de dispositivos móviles.

Otra de las características que cabe mencionar, es el desarrollo de una librería de gráficos 2D nativa, que ayudará a asumir el objetivo general del sistema, la rapidez y fluidez de este, o en términos más técnicos, la optimización de los recursos del dispositivo en su grado máximo.

Para el uso de gráficos 3D, se confió inicialmente en la especificación OpenGL 1.0, la cual se ha ido actualizando de forma conjunta al sistema operativo en los 5 años de existencia con los que ya cuenta.

[12] Máquina virtual utilizada por la plataforma android para ejecutar sus aplicaciones.

Android incluyó también un motor de bases de datos adaptado a dispositivos móviles, en ese caso el motor elegido es SQLite[13], que permite el almacenamiento de datos de forma local, lo que puede suponer una limitación, pero aporta gran sencillez de cara al desarrollador para usarla desde sus aplicaciones.
Dado el crecimiento de los dispositivos móviles y el gran número de capacidades que nos ofrecían, el desarrollo del sistema incluyó soporte para multitud de archivos multimedia, como podrían ser MP3, JPEG, MPEG4, etc., el cual como la mayoría de las características, se ha ido incrementando con cada una de las actualizaciones del sistema.

Otra característica importante era ofrecer control para todo el hardware adicional que estaban incluyendo los fabricantes en los dispositivos móviles, como serían acelerómetros, GPS, cámaras, 3G, Bluetooth, etc., además por supuesto de la telefonía GSM[14].

Por último Android creo un SDK[15] completo, con debugger y un plugin IDE[16] para el entorno de desarrollo Eclipse, facilitando así a los desarrolladores introducirse en este nuevo sistema, consiguiendo así una gran comunidad de investigadores que contribuyeran a la evolución del sistema.

Figura 1: Logotipo Android

[13] Sistema reducido de gestión de bases de datos relacional.
[14] Sistema global para las comunicaciones móviles establecido como estándar, libre de cargos, para la telefonía móvil digital.
[15] Software Development Kit, conjunto de herramientas necesario para el desarrollo de aplicaciones para una plataforma o sistema operativo concreto.
[16] Integrated Development Environment, programa informático compuesto por un conjunto de herramientas de programación.

2.3 Estructura de Mercado

Actualmente podemos decir que Google ha asumido su objetivo con Android, que era liderar el mercado de los smartphones, frente a sus mayores competidores, Apple y Blackberry.

Todos estos dispositivos comparten una gran serie de características, como serías pantallas multitáctiles, soporte para gran número de archivos multimedia, conectividad inalámbrica de todo tipo, como sería 3G, Bluetooth, Wifi, etc.

Entre los modelos de Apple actualmente podemos destacar el iPhone 4S, su última innovación junto con el sistema operativo iOS 5, en el que han incluido el procesador de doble núcleo conocido como A5, que es capaz de funcionar a una frecuencia de 1,2 GHz, junto a 1 GB de memoria RAM y gran incremento de hardware adicional.

Blackberry acaba de irrumpir con su último terminal la blackberry bold 9800 acompañado de su último sistema operativo el QNX, que junto a su procesador de único núcleo con una frecuencia de 1GHz, 512Mb de memoria RAM, y su completo y funcional teclado QWERTY, ejerce cierta influencia en el mercado.

Google acaba de lanzar al mercado su último terminal patrocinado, que en este caso ha sido fabricado por Samsung, y conocido comercialmente como Galaxy Nexus, y la versión 4.0 de Android, el cual hace gala de un procesador de doble núcleo a 1.2GHz y un 1GB de RAM además de un gran número de hardware adicional.

Como podemos ver, la mayoría de las compañías en términos de hardware, nos ofrecen cosas muy similares, siendo la mayor diferencia entre unas y otras el sistema operativo que son capaces de ofrecernos, y la experiencia de usuario que son capaces de conseguir.

El siguiente gráfico, nos ayuda a hacernos una idea más aproximada de la implantación que tiene actualmente Android en el mercado actual.

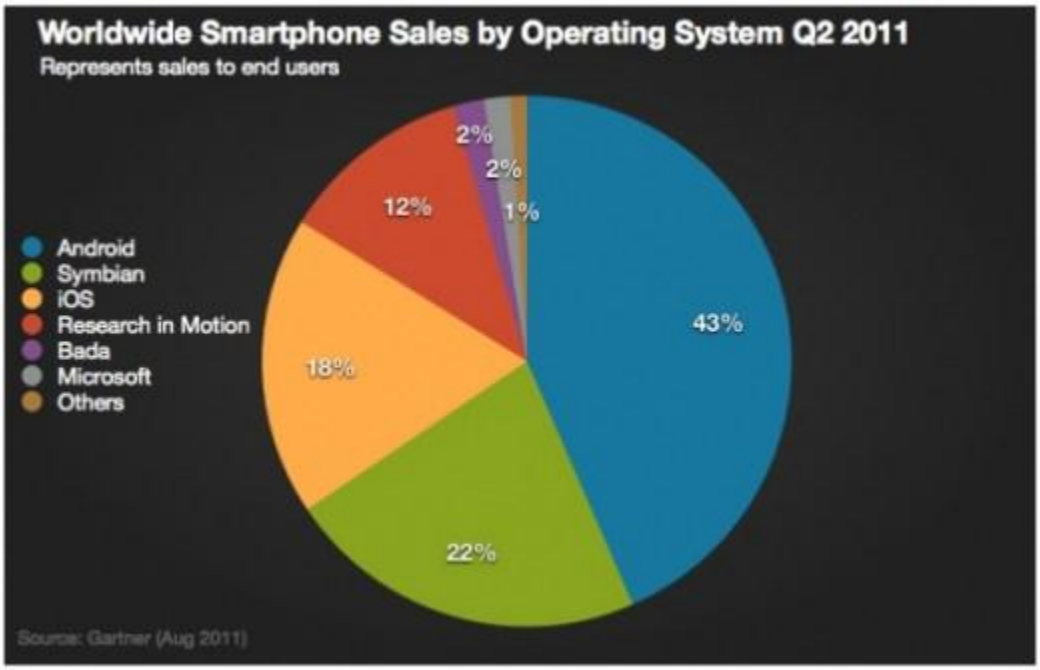


Figura 2: Estimación de la Población de Smartphones

2.4 Arquitectura del Sistema

De igual forma que la gran mayoría de los sistemas operativos, y dado que está basado en linux, adopta la arquitectura de este, es decir, una serie de capas, en las que se organizan cada uno de los componentes del sistema.

En la capa más profunda del sistema encontramos el kernel, la base del sistema operativo, en la capa inmediatamente superior encontramos las librerías del sistema, entre las que está contenido el Android Runtime, en la siguiente capa encontramos el Application Framework y finalmente en la capa superior, que es con la que interactúa el usuario, finalmente encontramos las aplicaciones.

El siguiente esquema, nos da una idea detallada de la arquitectura del sistema operativo Android.

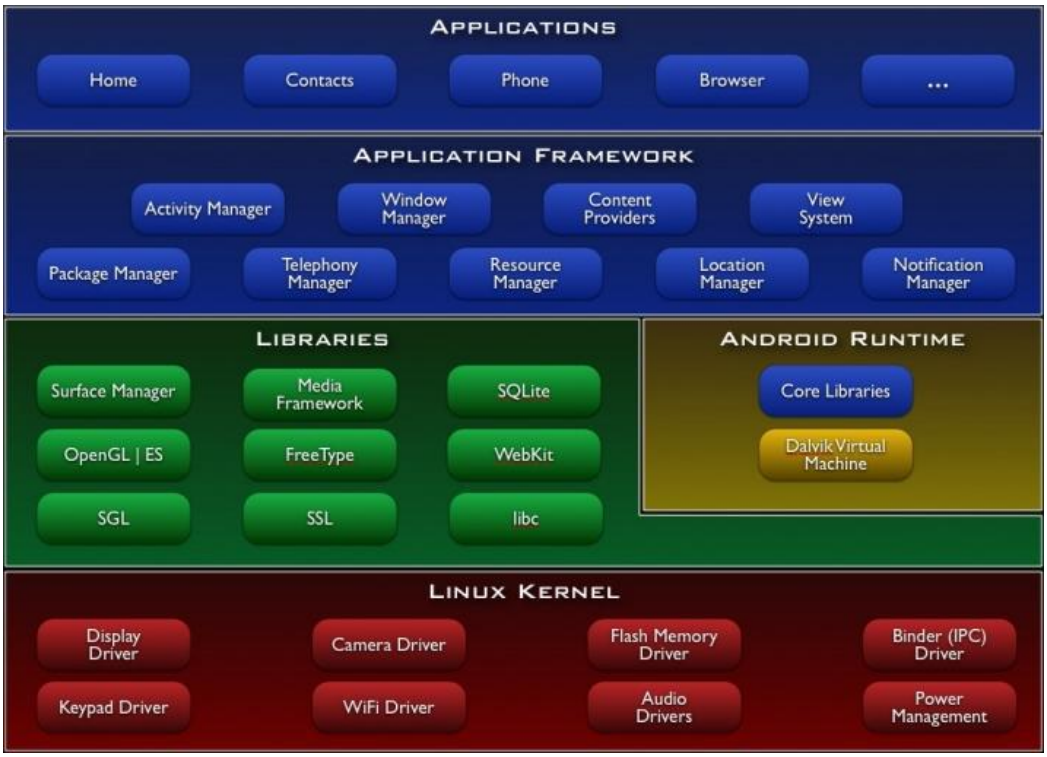


Figura 3: Estructura Interna del Sistema Operativo Android

2.4.1 Aplicaciones

Cada una de las aplicaciones desarrolladas sobre la plataforma Android, incluirán como base un cliente de email (correo electrónico), calendario, gestor de SMS's y una serie de API's. Todas estas aplicaciones comparten un framework común y están escritas en lenguaje JAVA.

2.4.2 Esqueleto de una Aplicación

Dado que Android es un sistema de código libre, todos los desarrolladores tienen acceso al código fuente de las aplicaciones básicas del sistema, Google ha dispuesto esto de esta forma para evitar el desarrollo de multitud de componentes que respondan a los mismos eventos o acciones.

Esto proporciona a los programadores, la posibilidad de que estos componentes sean modificados o usados de forma completamente libre por estos, permitiendo a estos llevar a cabo sus desarrollos de forma mucho más rápida y sencilla, ya que no tienen por qué iniciar sus aplicaciones desde cero en todos los casos.

2.4.3 Librerías

Android nos proporciona de forma nativa una serie de librerías que podemos usar de forma sencilla, desde cada una de nuestras aplicaciones para integrar gran cantidad de funcionalidades a nuestras aplicaciones de forma sencilla.

Entre estas librerías, podemos encontrar librerías SSL[17] para el soporte de certificados y seguridad, SQLite que nos permite gestionar el acceso a la base de datos local, OpenGL 3D para la integración de gráficos en tres dimensiones en la aplicación, WebKit para la optimización de la navegación de páginas web, el Media Framework para gestionar la gran cantidad de archivos multimedia que permite reproducir el sistema de forma nativa y el Surface Manager, que nos permite gestionar las diferentes pantallas en interfaces que podemos integrar en las aplicaciones, entre otras muchas.

[17] Secure Socket Layer, protocolos criptográficos estándar que se encargan de proporcionar comunicaciones seguras a través de una red.

2.4.4 Runtime de Android

Android incluye en su sistema un conjunto de librerías básicas que proporcionan al sistema la mayoría de funcionalidades básicas disponible en el núcleo del lenguaje de programación Java.

Como se ha mencionado con anterioridad en este documento, cada aplicación Android ejecuta su propio proceso, con su propia instancia de la máquina virtual Dalvik, la cual ha sido desarrollada de forma que un dispositivo sea capaz de ejecutar varias máquinas virtuales simultáneamente de forma eficiente. Para conseguir esta eficiencia, la máquina virtual inicia archivos ejecutables, con la extensión .Dex, cuyo formato esta optimizado para un consumo de memoria mínimo.

La máquina virtual está basada en registros, y ejecuta una serie de clases compiladas por un compilador Java que se han convertido al formato .Dex mediante el uso de la herramienta incluida "dx". Además esta máquina virtual se basa en el kernel de Linux para ofrecer las funcionalidades subyacentes, como podrían ser la gestión de memoria o el threading[18] a bajo nivel.

[18] Subproceso creado por el sistema operativo destinado a dividir procesos mayores en pequeñas partes que faciliten su ejecución.

2.5 Máquina Virtual Dalvik

La máquina virtual Dalvik, es una máquina virtual intérprete que ejecuta archivos en el formato .dex, Dalvik Executable, el cual como ya se ha hecho referencia anteriormente, esta optimizado para el almacenamiento eficiente y ejecución mapeable en memoria.

El objetivo primordial de esta máquina virtual, es el mismo que el de la gran mayoría de estas, permitir que el código fuente sea compilado a un bytecode de forma independiente al dispositivo en el que se vaya a ejecutar finalmente, encargándose ella misma de interpretarlo y ejecutar el programa.

El motivo principal por el cual se optó por esta máquina virtual antes que la de Java, era la extrema necesidad de optimizar recursos y enfocar el funcionamiento de los programas hacia dispositivos en los que la memoria, el procesador y el almacenamiento son escasos.

Otra característica importante de la máquina Dalvik, es el hecho de haber sido optimizada para que múltiples instancias de ella puedan funcionar de forma simultánea con un bajo impacto en el rendimiento de la memoria del dispositivo. El objetivo de este diseño, es proteger las aplicaciones, de forma que el cierre forzado de alguna de ellas no afecte al funcionamiento del resto de aplicaciones.

La diferencia básica entre la máquina virtual Dalvik y la máquina Java tradicional, es que esta última se basa en el uso de pilas, y la Dalvik usa registros, ya que los teléfonos móviles están optimizados para la ejecución basada en los mismos.

A pesar de que el lenguaje usado para el desarrollo de aplicaciones para android es Java, el bytecode de Java, no es ejecutable bajo android, de igual forma que las librerías de java usadas por Android, que son ligeramente diferentes a las usadas por el Java Standar(JAVA SE), guardando eso sí ciertas características en común.

El uso de la máquina virtual Dalvik, permite reducir considerablemente el tamaño de las aplicaciones, lo que es importante dado que el almacenamiento en este tipo de dispositivos es escaso, para conseguir esto lo que hace es buscar información repetida en diversas clases y reutilizarla.

El conocido como “Garbage Collector” de Java, usado para limpiar del espacio de memoria objetos que ya no se usar por nuestros programas, ha sido perfeccionado en Android, para mantener siempre el máximo espacio posible en la memoria, ayudando así a la optimización del sistema.

Para llevar más aún esta optimización Android usa de forma intensiva el lenguaje XML[19] para la generación de los ficheros que definen las interfaces gráficas, además de algunos otros datos, esto implica que estos archivos deben ser referenciados a lo hora de ejecutar la compilación, permitiendo así que su conversión a bytecode pueda mejorar el rendimiento de nuestras aplicaciones.

[19] **XML**, siglas en inglés de ***eXtensible Markup Language*** ('lenguaje de marcas extensible'), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C).

2.6 Partes de una aplicación

En este apartado se definen y explican los componentes básicos que forman cada una de las aplicaciones desarrolladas para la plataforma android. No todas las aplicaciones requieren todos estos componentes para ser funcionales, pero si la combinación de algunos de ellos.

En caso de querer usar alguno de estos componentes, deberemos especificarlo en el archivo AndroidManifest.xml, el cual especificaremos más adelante en el documento.

2.6.1 Activity

Este componente representa una única pantalla de la aplicación y se encarga de relacionarla directamente con una interfaz con la cual el usuario podrá interactuar. De entre las diversas actividades que pueden formar una aplicación, una de ellas se marcará como actividad principal o "Main", esta será la actividad encargada de iniciar la aplicación cuando el usuario la lancé desde el menú del dispositivo.

De igual manera una actividad puede iniciar otras actividades con el objetivo de componer la aplicación de diversas pantallas o interfaces con las que poder interactuar con el usuario, ya sea mostrándole datos que la aplicación ha generado, obteniendo datos que el usuario nos facilite mediante su interacción con esta.

2.6.2 Services

Un servicio es una tarea que el sistema ejecuta en Background[20], y que no tiene ninguna interfaz de usuario asociada, dado que el usuario en ningún momento llegará a interactuar de forma directa con este. Este componente se encarga de realizar operaciones de larga duración de forma que el usuario pueda seguir usando la aplicación mientras se generan los resultados esperados, o de realizar las tareas relacionadas con procesos remotos.

[20] Proceso o rutina de ejecución que se ejecuta en segundo plano, normalmente con una prioridad baja con el fin de no saturar el sistema operativo.

Algunos ejemplos de servicios podrían ser el reproductor multimedia o la escritura de ficheros en la memoria del dispositivo. Otro componente de la aplicación, como por ejemplo una actividad, se encargará de iniciar el servicio y dejarlo en Background para poder interactuar con él y obtener los resultados en cuanto estos estén disponibles.

2.6.3 Broadcast Recievers

Este componente recibe y reacciona con otros eventos de tipo broadcast que pueda producir la propia aplicación o el sistema en si, como podría ser el aviso de batería baja, SMS, etc.

2.6.4 Intents

Este componente es el encargado de activar los componentes mencionados hasta ahora, mediante este tipo de mensajes asíncronos. Este componente nos permite enviar mensajes a todo el sistema, a un Activity o a un servicio concreto indicando en este la acción que se quiere llevar a cabo. El propio sistema se encargará de decidir quien realizará la acción que se ha solicitado en el Intent.

2.6.5 Content Providers

Este componente nos permitirá acceder a los datos que queremos usar en nuestra aplicación desde las diferentes fuentes de datos que nos proporciona Android. Puede almacenar los datos en el sistema de ficheros del sistema, en la base de datos SQLite de la aplicación, una web o cualquiera de los otros lugares de almacenamiento definidos anteriormente.

A través de este tipo de componente podemos consultar y modificar los datos, en caso de que el Provider usado lo permita.

2.6.6 Layout

Uno de los puntos más importantes en el diseño de aplicaciones para dispositivos móviles es la interfaz de usuario, ya que será aquello que llame la atención de los potenciales compradores y lo que en definitiva hará que estos se decidan a usar nuestra aplicación.

Actualmente el diseño de interfaces gráficas es el punto más débil de Android, ya que no dispone de una entorno de diseño completo, sino que es una pequeña parte del plugin que integramos en Eclipse, debido a esta incidencia es necesario conocer el lenguaje XML para poder diseñar interfaces atractivas.

La interfaces en Android se definen sobre Layout's, que son ficheros XML en los que definimos la posición, la forma y una serie de propiedades de cada uno de los elementos gráficos que formarán parta de nuestra interfaz, mediante una serie de tags xml, que un compilador se encarga de traducir en posiciones sobre la pantalla.

Como hemos comentado con anterioridad una misma aplicación puede tener más de un Layout, de hecho lo más habitual es que tengan más de uno, ya que cada Activity tiene su propia interfaz, y por lo tanto su propio layout.

2.6.7 AndroidManifest.xml

Cada aplicación Android que desarrollemos debe tener un archivo AndroidManifest.xml, exactamente con este nombre, en su directorio raíz. Este fichero presenta información esencial sobre la aplicación al sistema, información que el sistema es imprescindible que tenga para poder ejecutar la aplicación. Algunas de las cosas que el manifest se encarga de aportar son las siguientes:

- Indica el package de la aplicación, el cual se usa como identificador único de esta dentro del sistema.

– Definir cada uno de los componentes de la aplicación, las activities, servicios, broadcast recievers and content providers de los cuales se compone la aplicación. Además nombra las clases que implementan cada uno de estos componentes y publica sus propiedades. Estas declaraciones son usadas por el sistema para saber las capacidades de los componentes y en que momento deben ser iniciados.

– Determina que procesos se encargarán de acoger a los componentes de la aplicación.

– Indica que permisos debe tener la aplicación de cara a acceder a partes protegidas de la API e interactuar con otras aplicaciones.

– A su vez indica los permisos que deben tener otras aplicaciones para poder interactuar con los componentes que forman nuestra aplicación.

– Lista las clases que se encargan de proporcionar los perfiles y demás información de la ejecución de la aplicación. Estas declaraciones únicamente están presentes en el AndroidManifest.xml mientras la aplicación está siendo desarrollada y probada, y son retirados cuando la aplicación se publica.

– Se declara el nivel mínimo de la API que el dispositivo debe ser capaz de ejecutar para poder usar la aplicación.

– Lista las librerías con las que se debe vincular la aplicación para poder funcionar correctamente.

2.7 Ciclo de Vida de una Aplicación

Las aplicaciones Android, no tienen control de su ciclo de vida, ya que es el propio sistema el que se encarga de gestionar este para poder tomar la decisión que más convenga al rendimiento del sistema, y por tanto del dispositivo en que se ejecute, en todo momento.

Esto quiere decir que el ciclo de vida de una aplicación Android lo maneja el sistema basándose en las necesidades del usuario, los recursos disponibles, las acciones solicitadas, etc. En caso de tener una aplicación en funcionamiento que este consumiendo una gran cantidad de recursos del sistema, y decidimos arrancar una segunda aplicación, el sistema se encarga de comunicar a la primera aplicación, que ahora queda en segundo plano, que liberé todos los recursos que le sea posible, y en caso de ser necesario puede llegar a forzar su cierre.

En Android los recursos del sistema son normalmente limitados y es por este motivo por el que el sistema tiene más control sobre todas sus aplicaciones que cualquier otro sistema de escritorio habitual.

Como hemos comentado con anterioridad, en la mayoría de los casos, cada aplicación lanza su propio proceso de Linux. Es proceso se crea para la aplicación cuando la iniciamos y seguirá ejecutándose hasta que no sea necesario y el sistema solicite los recursos que está usando para otras aplicaciones.

Para tomar todas están decisiones relacionadas con el ciclo de vida de las aplicaciones, Android ordena los procesos por importancia, tal y como se detalla a continuación:

2.7.1 Foreground process

Es el proceso que contiene la Actividad que se está mostrando en ese momento por pantalla, para lo que se ha invocado el método onResume(). Habitualmente el número de Foreground process que se ejecutan de forma simultánea en el sistema es muy limitado, y estos procesos serán finalizados únicamente si aun liberando todos los procesos posibles del sistema, la memoria de este sigue siendo escasa.

2.7.2 Visible process

Es un proceso que contiene un activity que es visible, pero que no está en primer plano, se ha creado debido a la ejecución del método onPause(). En ejemplo podría ser el estar leyendo un email y pulsar sobre una url de forma que el navegador web se inicia, pasando en ese momento la aplicación del mail de Foreground Process a Visible Process y siendo el navegador el nuevo Foreground Process. Este tipo de procesos tienen un nivel de prioridad alto para el sistema, por lo que este trata de evitar su cierre en la medida de lo posible.

2.7.3 Service Process

Este servicio es similar al que podría ejecutar cualquier distribución de Linux. Este tipo de proceso se encarga de hacer tareas en segundo plano que normalmente tienen una cierta importancia, por lo que el sistema nunca lo finalizará a no ser que sea imprescindible para poder mantener en ejecución todos los procesos Foreground y Visible. Este estado si inicia mediante la ejecución del método startService().

2.7.4 Background Process

Es un proceso que contiene un activity que actualmente no es visible para el usuario, y al que se llega después de la ejecución del método onStop(). Este tipo de proceso no tiene una importancia demasiado elevada, ya que puede ser alguna aplicación que se inició tiempo atrás y no se ha vuelto a usar, y que por lo tanto quedo en background.

Es por este motivo por el cual es importante liberar todos los recursos que nos sea posible cuando nuestra aplicación base a background, evitando así que el sistema decida cerrarla porque hace un uso indebido de los recursos de este.

2.7.5 Empty Process

Es un proceso que no contiene nada, y que es usado por el sistema como caché en el momento en el que se crea un nuevo proceso. Es común la eliminación de este tipo de procesos por parte del sistema con el objetivo de liberar memoria para la ejecución de procesos con mayor prioridad.

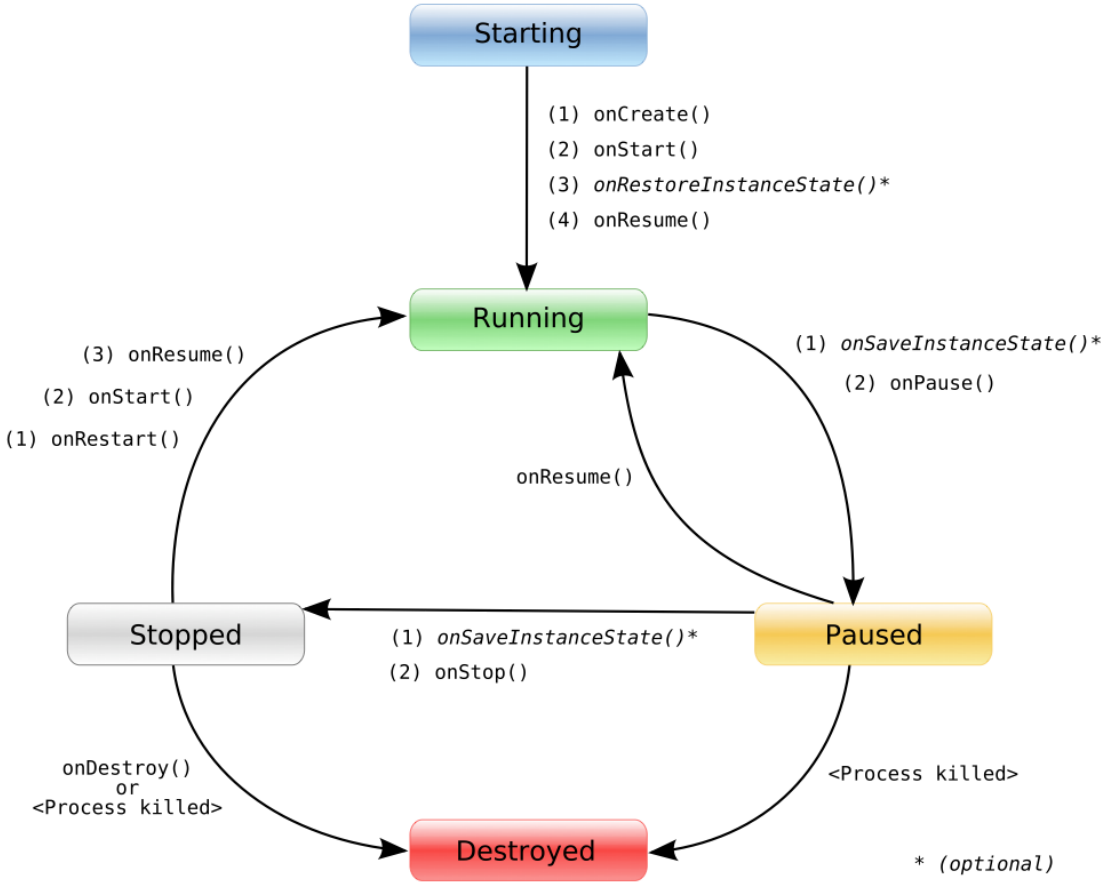


Figura 4: Esquema del Ciclo de Vida de una Aplicación Android

3. API ANDENGINE

3.1 Introducción

Una vez decidida la temática del proyecto, el primer paso a llevar a cabo era recabar toda la información posible sobre el desarrollo de videojuegos para la plataforma Android, después de un exhaustivo proceso de búsqueda, llegamos a la conclusión de que había dos forma de hacerlo, o bien usando las API's nativas de google o bien usando algunas API's externas.

Llevar a cabo el desarrollo mediante el uso de las API's nativas de Android, implicaba un gran número de horas de trabajo e investigación, sin tener la certeza de obtener un resultado final aceptable, dado que su complejidad es elevada y la documentación al respeto escasa.

El uso de API's externas facilitaba la tarea, ya que la documentación era mucho más extensa y el grado de complejidad de estas, se reducía de forma significativa. De forma que se decide hacer uso de alguna API orientada al diseño de videojuegos.

Era una requisito indispensable que la API elegida fuera OpenSource, por lo que después de una ardua tarea de investigación la escogida fue la librería AndEngine, ya que cumplía con todos los requisitos previamente establecidos, nos proporcionaba gran cantidad de funcionalidades relacionadas directamente con el desarrollo de videojuegos, y sobretodo es de código libre.

En el momento de llevar a cabo el desarrollo de videojuegos, hay una serie de procesos que pueden llegar a ser realmente complejos de desarrollar, como podrían ser la carga de imágenes o el motor de físicas, y es en estos puntos en donde AndEngine, resulta extremadamente funcional.

En los siguientes apartados detallamos las principales funcionalidades que nos ofrece esta librería.

3.2 Actividad Principal

Es parte básica del desarrollo de cualquier juego, una actividad que se encarga de gestionar todo el ciclo de vida del juego, sistema de puntuación, temporización en caso de ser necesaria, avance de los mapas, etc., en resumen todo aquello que hace que la experiencia de juego del usuario goce de cierta fluidez.

Es en este punto en el que usamos la llamada BaseGameActivity del AndEngine, que contiene las clases principales del flujo de juego. Para poder controlar este flujo, debemos crear una actividad que derive de BaseGameActivity y extender la implementación de sus métodos principales para adaptarlos a nuestras necesidades.

Siendo algunos de los eventos más típicos que deberemos controlar, el arranque del juego, la pantalla de inicio, la selección de nivel, personajes y demás configuraciones, en caso de que estás puedan llegar a existir, pausar el juego, etc.

3.3 Motor

El motor es la pieza encargada de hacer que el juego pueda llegar a funcionar. Dispone de un hilo de ejecución que refresca la pantalla, es decir aquello que el usuario está viendo, cada un intervalo de ciertos milisegundos, previamente definidos por el programador del juego.

En cada tick se encarga de sincronizar los refrescos de la pantalla, con la nueva situación de todos los elementos que forma parte de ella, y en definitiva de actualizar la escena.

3.4 Escenas

La escena es el contenedor de todos y cada uno de los sprites que se deben visualizar por pantalla, y para esto AndEngine nos facilita una clase, con una serie de métodos base que debemos completar para adaptarlos a las necesidades de nuestro luego.

Las escenas pueden estar formadas por diferentes capas y entidades, en las cuales podemos diferencias entre mapas, sistemas de puntuación e información al jugador, objetos con los que interactuar, personajes del juegos, etc.

3.5 Texturas

Las texturas son imágenes en memoria. Estas imágenes serán utilizadas para visualizar sprites o fondos, con los que componer la escena de nuestro juego. Es por esto que en el momento en el que se inicia el desarrollo de un juego, por una parte se gestiona la información lógica de los objetos y por otra parte el aspecto visual que finalmente tendrá, o lo que es lo mismo, las texturas que mostraremos.

3.6 Sprites

Los sprites son objetos que se visualizarán en el juego y en la mayoría de los casos serán interactivos o incluso animados. Existen diversos tipos de sprites, tiles en el que los juegos tienen forma de matriz, animados en los que el sprite se compone de varias imágenes o fotogramas, que se irán intercambiando a lo largo de la acción del juego para simular un movimiento o animación, o background los que serán usados para dibujar fondos.

3.7 Físicas y Colisiones

Adicionalmente disponemos de las funcionalidades necesarias para detectar colisiones e incluso aplicarles algunos efectos de física, mediante los cuales simulamos movimientos provocados por la interacción de diferentes elementos del juego, como serían el no poder atravesar una pared, o detectar que una bala ha impactado en un enemigo, provocando así nuestro victoria, consiguiendo así comportamientos mucho más reales de los objetos durante la acción del juego.

Para poder llevar a cabo toda esta interacción entre objetos, que como hemos comentado anteriormente tiene una complejidad de desarrollo bastante elevada, se usa una extensión basada en el motor de físicas de código libre Box2D, al cual le dedicamos todo un capítulo más adelante en este documento.

3.8 Música y Efectos de Sonido

Otra característica de este framework es la posibilidad de reproducir algunos tipos de ficheros de sonido. Permitiéndonos controlar ciertas características de la reproducción como serían subir y bajar el volumen, repetir el sonido, saber si está siendo reproducido en un momento concreto, escoger un punto concreto de la pista de audio, entre otras funcionas adicionales.

4. Box2D

4.1 Introducción

Box2D es un motor físico de código abierto creado principalmente para juegos. Es puramente un motor 2D, a pesar de esto, gracias a una evolución, ahora nos permite manejar polígonos convexos y algunas otras formas.

El motor Box2D, es una librería de simulación, de motor rígido, dirigida al desarrollo de videojuegos. Es usada por programadores de juegos, para hacer que sus objetos, se muevan de formas realistas, permitiendo esto llevar los juegos a una dimensión más interactiva.

4.2 Arquitectura, diseño y estructura

El motor Box2D trabajo con algunos objetos fundamentales, los cuales definiremos a continuación.
Objetos fundamentales del núcleo del motor.

- Shape, este objeto, nos permite definir objetos geométricos en 2D, como podrían ser un círculo o cualquier polígono.
- Rigid Body, esté objeto, nos permite definir un trozo de materia, un objeto, que es tan rígido, que la distancia entre dos partes de materia, en el bloque, es completamente constante.
- Fixture, este objeto, nos permite definir, un accesorio que se une a un cuerpo, como podría ser un Rigid Body, y agrega las propiedades del material, tales como la densidad, la fricción o la restitución.
- Constraint, este objeto, nos permite definir una conexión física que borra los grados de libertas de las formas. En 2D las formas tienen 3 grados de libertad.
- Contact Constraint, este objeto, es un constraint especialmente diseñado para prevenir la penetración de cuerpos rígidos, simulando así fricción y restitución. Este tipo de objeto es creado de forma automática por el motor.
- Joint, este objeto, es un constraint que nos permite tener dos o más cuerpos juntos. El motor Box2D soporta varios tipos de uniones, revoluto, prismáticos, a distancia y algunos otros más.
- Joint limit, este objeto, nos permite restringir el rango de movimiento de dos o más cuerpos juntos.

- Joint motor, este objeto, nos permite controlar el movimiento de conexión de cuerpos de acuerdo a los grados de libertas de los joint.

– World, este objeto, es una colección de cuerpos, accesorios y constraints que interacción entre ellos. Box2D soporta la creación de múltiples mundos, aunque esto no habitual o deseable.

–

La estructura de Box2D, se compone básicamente de 3 módulos, Common, Collision and Dynamics. El módulo de Common contiene el código de localización, matemáticas y ajustes. El módulo de Collision define las formas, la fase-amplia y las funciones y consultas de colisión. El módulo de Dynamics nos provee la simulación del mundo, cuerpos, accesorios y joints.

Las relaciones que existen entre estos tres módulos se definen de la siguiente forma. El módulo Common, tiene visión de las funcionalidades de los módulos Collision y Dynamics.

El módulo Collision, únicamente puede ver las funciones del módulo Dynamics. El módulo Dynamics no puede acceder a las funciones de ninguno de los otros dos módulos.

La estructura del motor, trabaja con números en coma flotante, dado que para permitir un buen desempeño del Box2D, debemos permitir algunas tolerancias en las definiciones.

Estas tolerancias han sido adaptadas para que funcionen correctamente con las unidades metros-kilogramo-segundo, MKS. Concretamente, el motor Box2D, ha sido adaptado para que funcione correctamente, con objetos que se desplazan entre los 0,1 y los 10 metros.

Esto quiere decir que tendremos que usar algún tipo de escalado cuando se procesa el entorno y los objetos. El banco de pruebas de Box2D, usa para este fin, una ventana de OpenGL.

En caso de manejar objetos estáticos, estos pueden ser de hasta un total de 50 metros, pudiendo ser manejados sin demasiados inconvenientes.

No usamos como unidad los píxeles, cosa que sería cómoda en un motor de física 2D, como es Box2D, dado que esto llevaría a una simulación excesiva y probablemente a un comportamiento extraño por parte del motor.

Box2D utiliza radianes para los ángulos. La rotación del cuerpo se almacena en radianes y puede aumentar sin límites. Debemos considerar la posibilidad de normalizar el ángulo del cuerpo si el valor de este ángulo es demasiado grande.

4.3 Funcionalidades

Pasaremos a especificar las funcionalidades del motor, divididas en los módulos que lo componen. En primer lugar explicaremos las funcionalidades del módulo Common, este módulo contiene los ajustes, la gestión de memoria y el vector math.

- Ajustes: la cabecera b2Settings.h contiene:

  - Tipos como int32 y float32: box2D define varios tipos de datos como serían int32, float32, int8, etc., para hacer más fácil determinar el tamaño de las estructuras.
  - Constantes: Box2D define algunas constantes, todas ellas están documentadas en b2Settings.h, normalmente no es necesario ajustarlas. Box2D usa coma flotante para la colisión y simulación.
  - Envoltorios de asignación: el fichero de configuración define b2Alloc y b2Free para grandes asignaciones. Estas llamadas pueden ser enviadas a nuestro propio sistema de gestión de memoria.
  - El número de versión: la estructura b2Version contiene la versión actual y la podemos solicitar en tiempo de ejecución.
  - Funciones de fricción y restitución: estas serán almacenadas en el fichero de configuración únicamente en caso de que queramos personalizarlas.

- Gestión de memoria: Box2D suele asignar un gran número de objetos pequeños, de entre 50 y 300 bytes. El uso de las variables del sistema a través de malloc para la gestión de los objetos pequeños es altamente ineficiente y puede ocasionar fragmentación.

La solución que ofrece Box2D para este problema, es utilizar un factor de imputación de objeto pequeño, SOA, el b2BlockAllocator. La SOA mantiene una serie de contenedores ampliables de diferentes tamaños. Cuando se hace una solicitud de memoria, la SOA devuelve el bloque de memoria que mejor se adapte al tamaño solicitado. Cuando un bloque se libera, se devuelve al contenedor. Ambas operaciones son rápidas.

Para esta gestión el usuario debe asignar un b2World. La clase proporciona creadores de cuerpos, accesorios y articulaciones. Esto permite a Box2D utilizar la SOA y ocultar los detalles menos agradables al usuario.

La ejecución de un step, Box2D necesita un pequeño espacio de memoria del área de trabajo temporal. Con este fin, se usa una pila del asignador b2StackAllocator, con el fin de evitar las asignaciones innecesarias, no debemos interactuar con la pila.

- Math: Box2D incluye en vector simple y pequeño del módulo matriz. Este ha sido diseñado para satisfacer las necesidades internas de Box2D. Todos los miembros del módulo están expuestos de forma pública, con el fin de que puedan ser usados desde la aplicación.

Ahora pasamos a explicar las funcionalidades del módulo Collision, este módulo contiene las formas y las funciones para poder operar con estas. Además contiene un árbol dinámico y la fase de onda para la aceleración del procesado de colisiones de sistemas grandes.

- Shapes: Box2D Shapes implementa la clase de base b2Shape, que define las siguientes funciones:
    - Prueba de un punto de superposición con la forma.
    - Realizar un rayo lanzado contra la forma.
    - Calcular la AABB de la forma.
    - Calcular las propiedades de forma de la masa.
    - Circle Shapes
    - Polygon Shapes
    - Shape Point Test
    - Shape Ray Cast

El módulo Collision contiene algunas funciones bilaterales que computa algunos resultados a partir de dos formas, estas funciones son:

- Contact mainfolds, esta función nos proporciona los puntos de contacto entre dos formas superpuestas.

- Distance, la función b2Distance puede ser usada para calcular la distancia entre dos formas, esta función necesita que las formas se conviertan a b2DistanceProxy.
- Time of impact: la función b2TimeOfImpact se usa para determinar el momento en que dos formas colisionaran. El uso principal de esta función es encontrar el túnel.

- Árbol dinámico: la clase b2DynamicTree es usada por el motor Box2D para organizar largas cadenas de formas de forma eficiente. La función opera en el eje alineado al cuadro delimitador de AABB con datos pasados por el usuario.

El árbol dinámico es un árbol jerárquico AABB, cada uno de los nodos del árbol contiene dos hijos. La estructura del árbol permite una eficiente consulta de la región.

- Fase de Onda: la clase b2BroadPhase reduce la carga de trabajo del cálculo de puntos de contacto mediante el uso de un árbol dinámico, reduciendo así considerablemente el número de llamadas a la función.

Por último definimos las funcionalidades que nos ofrece el módulo Dynamics, que son las descritas a continuación. Este módulo es la parte más compleja del motor, y es la parte con la que más interactuamos. Este módulo contiene:

- Shape Fixture Class: debemos tener en cuenta que la clase no sabe nada de cuerpos y puede ser usada de forma independiente, por lo que esta nos permite fijar las formas de los cuerpos. Los accesorios contienen lo siguiente:
    - Single Shape: usamos esta parte para la creación y almacenamiento de datos de la forma.
    - Density, friction and restituion
        - Density: la densidad es usada para computar las propiedades de masa del accesorio padre.
        - Friction: la fricción es usada para hacer que los objetos se desplazan a lo largo de los otros de forma realista.
        - Restitution: la restitución es usada para hacer que los objetos reboten.
    - Collision filtering flags: este es un sistema para prevenir la colisión entre figuras. Box2D soporta un total de 16 categorías diferentes de colisión.
    - Back pointer to parent body
    - User data
    - Sensor flag: usados para permitir a la lógica del juego saber que dos figuras se superponen pero no debe haber respuesta.

- Rigid Body Class: los cuerpos tienen una posición y una velocidad, podemos aplicar fuerzas, esfuerzos e impulsos. Existen tres tipos de cuerpos:

  - Static: los cuerpos estáticos no pueden moverse, haciéndonos creer que su masa es infinita. Estos no pueden colisionar con otros static bodies ni con kinematic bodies.
  - Kinematic: estos cuerpos se mueven bajo una simulación de acuerdo a su velocidad y no responden a fuerzas. Estos no puede colisionar con static u otros kinematic bodies.
  - Dynamic: estos son cuerpos completamente simulables, puede aplicársele fuerzas y puede colisionar con todos los otros bodies.

Para definir bodies debemos definir los siguientes miembros:

- Body Type: a escoger enter static, kinematic y dynamic.6
- Position and angle: la posición nos permite definir el punto original del body y el ángulo nos permite definir el ángulo del polígono.
- Damping: este parámetro es usado para reducir la velocidad del body.
- Sleep Parametres: parámetro usado para definir el tiempo durante el cual el body deberá estar quieto en caso de que la cpu este excedida.
- Fixed Rotation: nos permite definir una inercia rotacional sobre el body.
- Bullets: son usados para implementar el cd con static y dinamice bodies.
- Activa tion: nos permite definir que un body que ha sido creado no participe en la colisión.
- User Data

Para crear o destruir bodies, debemos usar la Body Factory, proporcionada por la clase World. Nos permite crear el body con un localizador eficiente y añadir este a la estructura de datos del mundo.

Una vez creados los bodies, estos pueden ser usados en tiempo de ejecución y podemos querer modificar o consultar algunos parámetros como serían Mass Data, Statu Información and Position and Velocity

- Contact Class: objetos usados en el motor Box2D para manejar colisiones entre accesorios. La clase Contact nos permite crear y destruir objetos.

Para poder acceder a los contactos disponemos de diferentes vías, directamente desde el mundo o desde la estructura de los bodies.

Podemos recibir datos del contacto implementando el método b2ContactListener, que nos informará de los contactos que hayan sucedido.

Podemos filtrar aquellos eventos que no nos interesen que se den o que tengamos mayor interés por usar, mediante la implementación del método b2ContactFiltering.

- Joint Classes: los objetos de esta clase son usados para definir restricciones a los bodies respeto al mundo o respeto a otros bodies.

Para definir joints debemos tener en cuenta los siguientes puntos:

- Todos los joints están conectados entre dos bodies, siendo uno de estos de tipo static. Los joints entre static y kinematic a pesar de estar permitidos, no tienen efecto.
- Podemos definir flags para prevenir las colisiones entre nuestros bodies con los demás.
- La mayoría de definiciones de joints, requieren que se proporcionen algunos datos geométricos.

De igual forma que con los bodies, para crear joints debemos usar el Joint Factory, que se encargará de crearlos usando los métodos de la clase World.

Tenemos diferentes tipos de joints, los cuales definiremos a continuación:7

- Distance Joint: usado para decir que la distancia entre dos puntos en dos bodies debe ser constante.
- Revolute Joint: define que el conjunto de fuerzas de dos bodies de tener su punto de anclaje en común.
- Prismatic Joint: con este joint impedimos la rotación relativa entre dos bodies.
- Pulley Joint: usado para crear una polea idealizada, la polea conecta dos bodies a la tierra y entre sí, de manera que si uno sube el otro baja.
- Gear Joint: usado para la creación de artilugios mecánicos sofisticados
- Mouse Joint: usado para conducir un punto de un cuerpo hacia la posición actual del cursor del ratón.
- Line Joint: Usado para modelar un la rueda de un vehículo junto con la suspensión.
- Weld Joint: usado para tratar de limitar todo el movimiento relativo entre dos bodies.

- World Class: esta clase contiene los bodies y los joints anteriormente explicados. Es la encargada de manejar todos los aspectos de la simulación y permite las consultas asíncronas, como son las consultas AABB y los ray-casts.

Para crear un objeto de la Clase World, basta con indicar un vector de gravedades y un booleano indicando si los objetos pueden entrar o no en sleep.
Algunos de los parámetros que podemos manejar mediante esta clase, son los siguientes:

- Simulation: esta clase es usada para conducir toda la simulación, el usuario puede especificar el tiempo de paso, la velocidad y la posición del iterator de cuenta.
- Explorating: esta clase es el contenedor de bodies, contacts y joints, es posible para el usuario recorrer la lista de bodies, contacts y joints e iterar sobre estas.
- AABB Queries: este tipo de consultas son usadas para conocer todas las formas que hay dentro de una determinada área.
- Ray Casts: opción usada para determinar disparos y tiros parabólicos similares.
- Forces and Impulses: el usuario puede aplicar fuerzas, torsiones e impulsos a los cuerpos. Cuando aplicamos algunos de estos parámetros, proporcionamos al mundo el punto donde debe ser aplicado.
- Coordinate Transformations: esta clase contiene algunas utilidades que nos permiten transformar puntos y vectores entre el espacio local y el del mundo.
- Lists: es posible iterar sobre todos los accesorios de un cuerpo, mediante estas listas.

5. Servidor Multijugador

5.1 Introducción

Uno de los objetivos principales del proyecto es la integración del juego con un servidor multijugador, y para poder asumir este objetivo se inició un proceso de investigación, en el que se valoran una serie de opciones que pueden ofrecer las funcionalidades básicas necesarias para asumir el objetivo.

De entre todas las opciones, las dos soluciones que mejor se adaptan a nuestras necesidades son una combinación del servidor de bases de datos MySQL y Apache Tomcat, y en segundo lugar el servidor SmartFoxServer, en su última actualización 2X.

5.2 SmartFoxServer

SmartFoxServer es una plataforma completa para el desarrollo rápido y sencillo de aplicaciones o juegos multijugador. Ha sido desarrollado pensando en la simplicidad, permitiendo así a los desarrolladores crear cualquier tipo de desarrollo de forma rápida.

Algunas de sus principales características son la sencillez de uso, por lo que se han encargado de eliminar todas las complejidades innecesarias para hacer el servidor más intuitivo, ágil y útil.

También es importante la versatilidad de la que hace gala, para lo que se han rediseñado partes de su arquitectura para que sea aún más flexible, permitiéndonos así diseñar juegos y aplicaciones con mayor comodidad y control.

El rendimiento que nos ofrece el servidor, es de primer nivel, y en la versión 2X de SmartFoxServer, con la actualización 3.0 de su motor, ofrece un alto rendimiento gracias a una serie de mejoras.

5.3 Conexión

Partiendo de la premisa de que el desarrollo del servidor no es parte de este proyecto, el estudio de este se limita a sus características más básicas, con tal de obtener los conocimientos básicos para poder establecer una conexión desde el juego con este.

Hemos añadido una clase adicional al desarrollo del juego en la que hemos implementado la conexión de nuestro juego con el servidor, la cual se establecerá ya sea mediante WiFi o haciendo uso de las conexiones de datos inalámbricas de las que disponen los dispositivos Android.

Para poder establecer la conexión es necesario saber que el servidor se inicia y se publica haciendo uso de la máquina virtual Java, sobre un servidor Unix, el cual nos proporciona una dirección IP mediante la cual podemos identificar el servidor e iniciar el protocolo de conexión.

6. Lamb Pastor, la aplicación

6.1 Interés

Actualmente tal y como comentamos en el apartado inicial de esta documento, el aumento de las redes sociales es muy considerable, lo que supone la incorporación de miles de usuarios diariamente a este tipo de páginas, lo que supone una fuente potencial de usuarios en incremento para nuestro juego.

Usando la idea inicial de poder conseguir ocupar esos pequeños espacios de tiempo en la vida cotidiana muchas de las personas de la sociedad, tendencia cada vez mayor al uso de las redes sociales, y el increíble aumento de la población de smartphones con capacidades para comunicarse con internet, se plantea la creación de un juego que reúna ambos conceptos.

De igual forma que todos los campos relacionados con las nuevas tecnologías, el desarrollo de videojuegos para smartphones está experimentando un fuerte crecimiento, demostrando ser una fórmula de gran éxito, como podemos ver en juegos como el famoso Angry Birds...

El proyecto que se propone nace a raíz del interés que despierta la plataforma Android y el desarrollo de videojuegos, ya que es un mercado con un mercado considerado exponencial. Para esto hemos creado una idea original, basándonos en esquemas de juegos rápidos.

Para eso partimos de la idea de un deporte tradicionalmente catalán, el Gos D'Atura, cuyo concepto es muy simple, tenemos un rebaño de ovejas que es liberado en el centro de un recinto cerrado y un perro que debe conseguir meterlas a todas ellas dentro de un cercado en el menor tiempo posible y de la manera más ordenada posible.

6.2 Estructura

En el contenido de este apartado explicaremos la estructura de nuestra aplicación, más concretamente juego, las partes que la componen y los elementos necesarios para su funcionamiento.

6.2.1 Esquema general

El esquema de nuestra aplicación parte de la base de un terminal Android, y a partir de aquí es escalable. Tenemos un terminal en el que podemos jugar de forma completamente funcional y sin ningún tipo de requerimiento adicional, contra una inteligencia artificial previamente programada.

Un segundo elemento sería el uso de las capacidades inalámbricas del dispositivo para obtener una conexión a internet, momento en el cual entre en juego la tercera parte del juego, el servidor mediante el cual entre en juego el apartado multijugador de nuestro juego.

Cuando un usuario desee iniciar la modalidad multijugador, se requerirá en primer lugar un registro, mediante un username en el servidor, en el que está alojado el host encargado de controlar toda la interacción online.

El segundo paso será entrar en alguno de los lobbys que ofrece este y finalmente dentro de una habitación (room), en la que habrá una serie de usuarios dispuestos a iniciar una nueva partida, en este punto es posible unirse a una partida propuesta por un usuario que ya se encuentre en la sala, o bien crear su propia partida y esperar a que algún otro usuario decida unirse a esta.

La comunicación entre el móvil y el servidor se realiza mediante una clase incluida en la aplicación que se encarga de sincronizar toda la comunicación del terminal con el servidor.

6.2.2 Clases de la Aplicación

El juego Lamb Pastor, está formado por un total de 9 clases, que son básicamente creación de escenarios, gestión de personajes y reproducción de audio y efectos de sonido. Mostramos inmediatamente debajo un esquema con todas las clases que conforman el bloque de nuestra aplicación, y la relación que existe entre cada una de ellas.

Bloque de Escenarios

En este sub-esquema podemos ver las clases que se encarga de cargar los escenarios sobre los que se lleva a cabo la acción del juego, además de la que se encarga de establecer los límites y objetos con los que pueden interactuar los personajes, y por último todos los elementos que proporcionan la información del avance del juego al usuario.

Bloque de Personajes

En este segundo bloque podemos encontrar las clases relacionas con la creación de los personajes y la gestión de sus movimientos. Están clases se encargan adicionalmente de manejar todas las físicas que conseguirán recrear de una manera muchos más fidedigna los choques, rebotes y demás efectos de los personajes con los límites y obstáculos previamente definidos.

Bloque Principal

En este último bloque encontraremos las clases que se encargan de coordinar los dos anteriores para seguir el ciclo de vida del juego, y permitir al usuario tener una experiencia de juego agradable. Encontramos en este bloque la clase más importante de toda la aplicación, la encargada de gestionar el motor de juego y por lo tanto de que el juego en general funcione.

6.3 Casos de Uso

A continuación trataremos de detallar todas las posibilidades de uso que tiene nuestro juego partiendo de la pantalla inicial, y haciendo uso de la interfaz de usuario para navegar a todas sus pantallas.

6.3.1 Selección de Juego

En primer lugar tenemos la pantalla de inicio del juego, en la que podemos escoger entre las dos modalidades que ofrece este, single player o multiplayer, en esta primera pantalla tenemos una opción adicional para llegar a la pantalla de configuración del juego, en la que podremos escoger los juegos de los que queremos gozar durante nuestra acción de juego.

Figura 5: Pantalla Inicio Juego

6.3.2 Juego Individual

Una de las modalidades que ofrece nuestro juego es la de jugar de forma local contra una inteligencia artificial previamente programada. En este modo será el propio motor del juego el encargado de simular un movimiento de las ovejas, de manera que estas huyan usando toda la amplitud del escenario, complicando, siempre dentro un límite razonable, la tarea del usuario de llevar estas dentro del corral.

Cuando se inicia el juego, se inicia automáticamente la cuenta atrás del tiempo máximo que el usuario tiene para conducir a las ovejas con su perro al interior del cercado. El objetivo es conducir a todo el rebaño al interior de este antes de que el tiempo se agote, resultando así cumplido el objetivo del juego y por lo tanto ganando la partida.

En este momento es cuando el usuario debe pulsar sobre el perro y empezar a arrastrarlo en dirección a la oveja que quiera guiar hacia el corral, en el momento en que el perro entre en contacto con la oveja, esta dejará de huir para pasar a seguir las instrucciones del perro, que serán ir en la dirección en que este la empuje.

Figura 6: Escenario de Juego

Cabe la posibilidad de que la oveja llegue a un punto en que no pueda moverse por la influencia del perro, momento en que el usuario puede hacer uso del botón de ladrido para sobresaltar la oveja y que esta vuelva a correr libremente hacia una posición más ventajosa para el jugador en la que volver a intentar guiarla al interior del cercado.

Pulsando sobre el botón hardware del dispositivo aparecerán las siguientes opciones, que nos permitirán en primer lugar concretar ciertas opciones del juego, o en caso de desearlo, finalizar la partida y volver a la pantalla principal

Figura 7: Menú Contextual

6.3.3 Juego Online

En esta segunda modalidad, no se iniciará directamente la acción del juego como el primer caso de uso, sino que deberemos ingresar en alguna de las salas del servidor, usando un username que nos permite identificarnos dentro de este, y crear una partida esperando a un segundo jugador.

En el momento en el que haya dos jugadores decididos a empezar una partida, las ovejas empezar a desplazarse, y los jugadores deberán meter tantas ovejas como les sea posible dentro de su cercado antes que su contrincante, resultando ganador aquel que consiga meter más ovejas dentro de su cercado al final de tiempo de juego establecido.

Disponemos del mismo menú contextual que en la modalidad individual, que nos permite realizar las mismas acciones que en este.

Figura 8: Escenario Juego Modalidad Multijugador

6.3.4 Configuración

En un último caso de uso, planteamos las opciones de configuración que nos proporciona nuestra aplicación, y que se mantendrán a lo largo de nuestra interacción con este, y que podremos modificar desde el menú contextual de este en todo momento, para poder adaptarlas a nuestras necesidades.

Las opciones que nos permite configurar son las relacionadas con los sonidos del juego, mediante dos CheckBox, podremos seleccionar si queremos disfrutar de los sonidos de los personajes y de la banda sonora escogida para el juego.

Figura 9: Pantalla de Configuración

7. Costes y posibles mejoras

7.1 Estudio económico

El coste total de la aplicación se desglosará a continuación en forma de tabla, con el fin de facilitar su compresión. Hay que tener en cuenta que una de las condiciones del proyecto era realizarlo de forma íntegra sobre plataformas Open Source, es decir con licencias gratuitas.

Los costes se reducen a los materiales necesarios para llevar a cabo el desarrollo y las pruebas necesarias, y el alojamiento del servidor encargado de gestionar las aplicaciones multijugador, lo que no es parte de este proyecto, y que por lo tanto, no se incluirá en este estudio.

Hardware	Precio Estimado
Ordenador(cumpliendo requisitos mínimos)	600€
Dispositivo Android(cumpliendo requisitos mínimos)	150€

Software	Versión	Licencia	Precio Estimado
Ubuntu	11.04	GPL	Gratuito
Eclipse	Indigo 3.7.1	GPL	Gratuito
Android SDK	R16	GPL	Gratuito
Eclipse Plugin ADT	15.0.1.v201111031820-219398	GPL	Gratuito
AndEngine		GPL	Gratuito
Gimp	2.6.11	GNU GPL	Gratuito
LibreOffice	3.3.4	LGPL	Gratuito
ArgoUML	0.34	GNU GPL	Gratuito

RR. HH.	Horas estimadas	Precio/Hora €	Precio Estimado
Diseñador	75	60	4500€
Programador	150	55	8250€

Concepto	Precio Estimado
Hardware	750€
Software	0€
RR.HH.	12750€
Coste Total: 13500€	

7.2 Mejoras y Ampliaciones

Partiendo de la limitación inicial de tiempo que teníamos para llevar a cabo el proyecto, existen diversas mejoras y ampliaciones que podrían incorporarse a la aplicación.

Una primera mejora que se podría considerar sería la introducción de nuevos mapas y escenarios sobre los que llevar a cabo la acción del juego, además de introducir nuevos perros con los que poder jugar.

Otra posible mejora sería introducir perfiles de usuario, que permitan a estos potenciar las habilidades de su perro haciéndolo evolucionar y por lo tanto resultándole cada vez más sencillo asumir el objetivo del juego.

Cabría también la posibilidad de permitir al usuario crear sus propios escenarios, ayudando así al crecimiento de la aplicación. De forma simultánea conseguimos también que el usuario se siente en contacto con la aplicación y que su uso de esta sea mucho más habitual.

Por último sería interesante revisar la eficiencia de la aplicación consiguiendo que la experiencia de juego resultara más fluida en dispositivos con hardware más limitados que los actuales.

8. Conclusiones

8.1 Objetivos alcanzados

Puede decirse que los objetivos definidos inicialmente han sido alcanzados en su totalidad con éxito. Hemos asumido el estudio de la estructura de Android, creando nuestra propia aplicación para conseguirlo, la que hace uso de muchas de las capacidades del sistema, como serían los eventos táctiles, librerías gráficas, conectividad inalámbrica, etc.

El principal objetivo del proyecto era entender el funcionamiento de cada una de las partes que conforman una aplicación Android. Realizando el estudio teórico detallado de cada una de estas partes para pasar luego a una aplicación práctica de estos y comprobar si se habían asumido los conceptos básicos.

Un objetivo adicional era el estudio y uso de la librería AndEngine para la creación del juego, para lo que hemos estudiado una serie de ejemplos de funcionamiento y la documentación proporcionada por esta en el momento de su obtención, para luego pasar a adaptarla a nuestras necesidades y encontrarnos con las limitaciones de este, haciendo la experiencia mucho más completa y enriquecedora.

Finalmente como se ha comentado, el estudio del protocolo de conexión con el servidor SmartFoxServer, que nos permite llevar a cabo la experiencia online, que era el objetivo primordial del proyecto.

9. Bibliografía

[1] Beginning Android 2. Murphy, Mark. Editorial Apress – 2010.

[2] Beginning Android Games. Zechner, Mario. Editorial Apress – 2011.

[3] Pro Android Games. Silva, Vladimir. Editorial Apress - 2010.

[4] Android Official Developer Program: [Consulta: 15/10/2011]
http://android.developer.com

[5] AngEngine Support: [Consulta: 15/09/2011]
http://andengine.org

[6] Foro especializado en Android: [Consulta: 17/09/2011]
http://www.android-spa.com

[7] Foro especializado en Android: [Consulta: 19/09/2011]
http://www.stackoverflow.com

[8] Wikipedia: [Consulta: 20/11/2011]
http://www.wikipedia.org

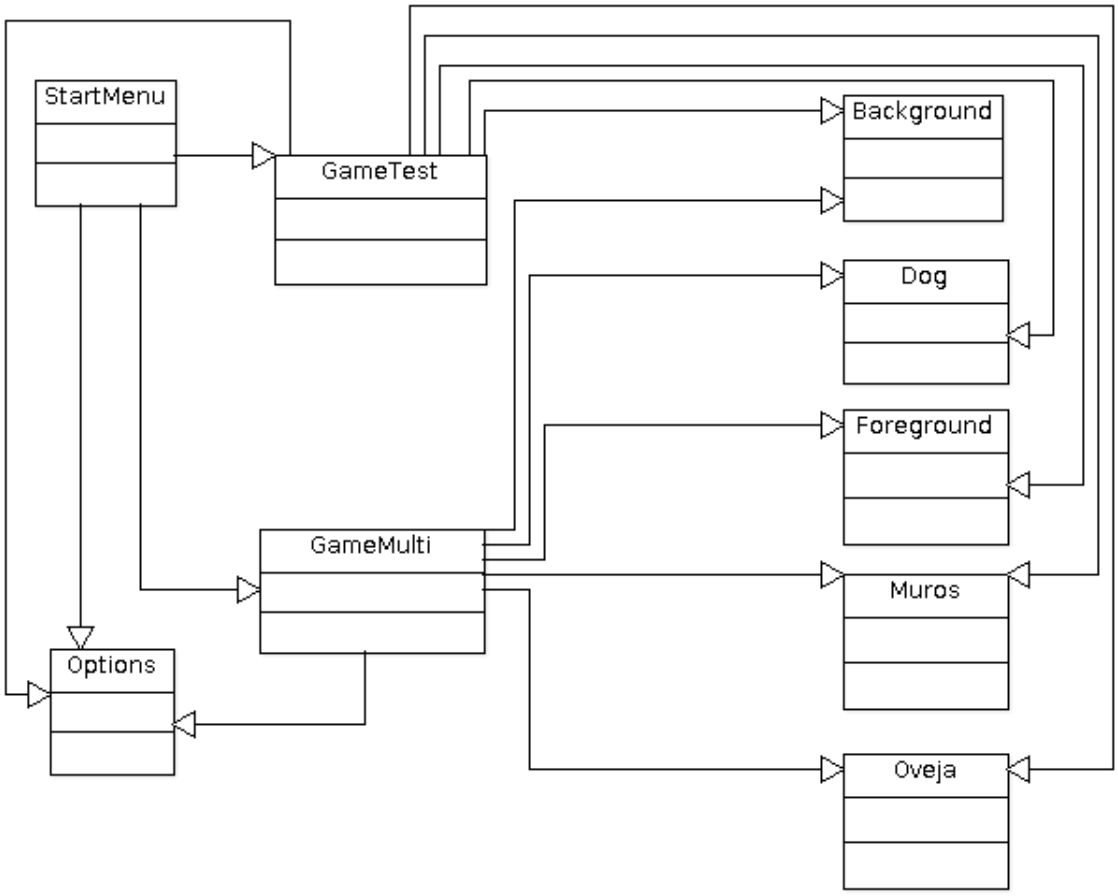
[9] Máquina Virtual Dalvik: [Consulta: 21/11/2011]
http://androideity.com/2011/07/07/la-maquina-virtual-dalvik/

[10] Box2d Official Site: [Consulta: 20/10/2011]
http://www.box2d.org

10. Anexos

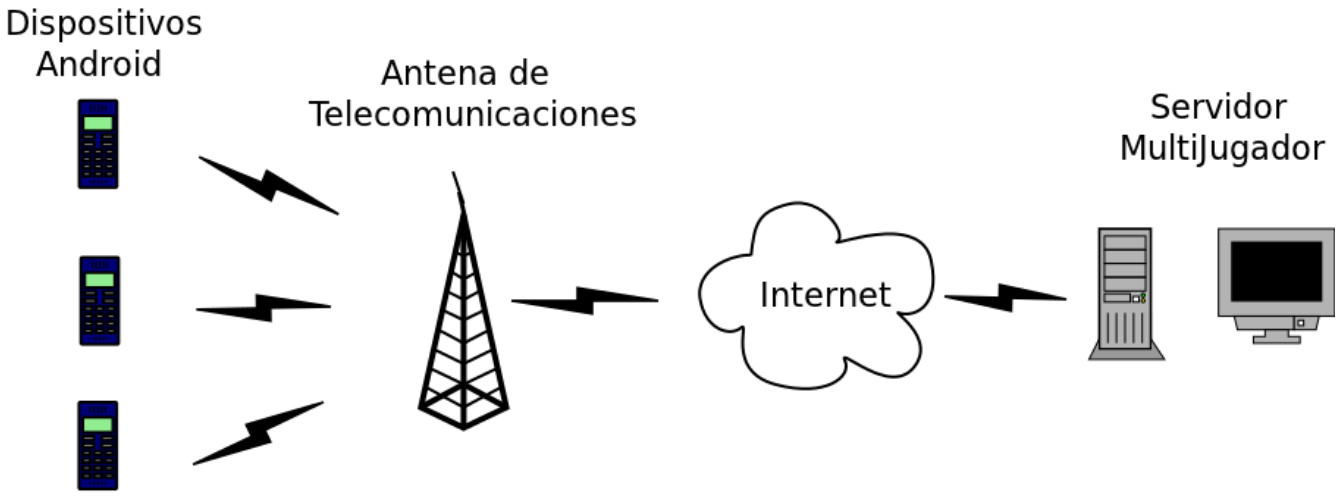
10.1 Anexo I: Diagrama de Clases

En el siguiente diagrama se detallan las relaciones que existen entre todas las clases que conforman el la aplicación final.



10.2Anexo II: Diagrama de Comunicaciones

En el siguiente diagrama detallamos todas las comunicaciones que es necesario establecer para el correcto funcionamiento de la aplicación en su modalidad multijugador.



10.3Anexo III: Imágenes de los componentes de la aplicación

A continuación se muestra una visión general de todos los componentes que forman parte del proyecto, para tener una visión más general de todas las tecnologías implicadas.